MUON $g - 2$

OFFLINE COMPUTING AND SOFTWARE MANUAL

v5_00_00 November 13, 2014 [gm2swdocs feature/markdownify 10fda9d] GM2-doc-1825

Contents

1

Getting started with gm2artexamples

This section is a short tutorial to show you quickly how to get started by,

- Logging in and selecting a release (the latest)
- Starting a development area
- Checking out code (`gm2artexamples`)
- Building it
- Testing
- Running
- Logging in again

For this tutorial, we'll use the `gm2artexamples` product.[1] This is a good product to use if you are getting started.

[1] We use the terms *product*, *project*, and *package* somewhat interchangeably. All of our products live on the Redmine server, http://redmine.fnal.gov

1.1 Logging in and selecting a release area

Fermilab has several interactive virtual machines for use by the Muon $g-2$ collaboration. See here for more information about how to log in. Our releases (libraries, executables) are served by CVMFS.[2] CVMFS is already mounted on the Fermilab interactive VMs. If you have a Mac, you can install CVMFS yourself by looking here, and then use your Mac to develop code.

[2] CVMFS is a system that serves application code and updates automatically when new files are released.

Once you've logged into the machine, you need to select a release area. You *always*[3] need to do this step everytime you log in. If you are on a Fermilab interactive VM (`gm2gpvm01`, `gm2gpvm02`, `gm2gpvm03`, `gm2gpvm04`), you select the release area by doing,

[3] You need to do this step everytime you log in because you can use different release areas for the same development area, say, for example, if CVMFS is down or you are sharing a directory between your Mac and a Linux system.

```
$ source /grid/fermiapp/gm2/setup # On gm2gpvm machine
```

Note that `$` is the shell prompt (don't type it in).

If you are on a Mac or another system with CVMFS OASIS installed, you do (and will see in response, which will be the same as the command above)

```
$ source /cvmfs/oasis.opensciencegrid.org/gm2/prod/g-2/setup # On Mac
```

```
g-2 software

--> To list gm2 releases, type
ups list -aK+ gm2

--> To use the latest release, do
setup gm2 v5_00_00 -q e6:prof

For more information, see https://cdcvs.fnal.gov/redmine/projects/g-2/wiki/ReleaseInformation
```

1.2 Starting a development area

Now that the release area is selected, you need to make a *development area*. The development area contains source code, build products, and a personal release area. You typically use a development area for a particular topic, such as adding a feature to the simulation or generating a plot for some study. You can have as many development areas as you want, but only one can be active at a time.

Make an empty directory and go there. If you are on a `gm2gpvm` machine, you should make an area in `/gm2/app/users/<YOUR_NAME>`.[4] You can put code in your home directory, but that has a small quota and you can easily use it all up. There is no quota on `/gm2/app`, but it is not backed up.

[4] If this directory does not exist, you can make it with the `mkdir` command.

```
$ mkdir /gm2/app/users/lyon/first-try  # On gm2gpvm
$ cd /gm2/app/users/lyon/first-try
```

If you are on your Mac, or some other machine, make the directory where you have room.

```
$ mkdir ~/Development/g-2/first-try # On Mac
$ cd ~/Development/g-2/first-try
```

Since you are starting out with a new area, you must choose a release. You should generally choose the latest, which will be specified in the output when you selected the release area. Just do what the command says,[5]

[5] `setup` is a *ups* command. UPS is our release and product management system.

```
$ setup gm2 v5_00_00 -q e6:prof
```

So here we are setting up g-2 release `v5_00_00` with the `e6:prof` qualifier. `e6` indicates the type of compiler we're using (in our case `gcc 4.9.1` with C++14 features turned on - this code is decided by the art team) and `prof` means we'll do a profile build. Profile builds are optimized and have debugging symbols turned on. We only use profile builds.

Now, you must create the development area. You will start using the `mrb` commands. *mrb* means "multi-repository build system" and is a build system used by Muon $g-2$, the art developers, and LBNF. You can get a list of `mrb` commands with (you don't have to type in the full path that you see below),

```
$ mrb -h
```

```
Usage /cvmfs/oasis.opensciencegrid.org/gm2/prod/external/mrb/v1_03_00_gm2/bin/mrb  [-h for help]"

  Tools ( for help on tool, do "/cvmfs/oasis.opensciencegrid.org/gm2/prod/external/mrb/v1_03_00_gm2/bin/mrb <tool> -h" )

    newDev (n)                 Start a new development area
    gitCheckout (g)            Clone a git repository
    svnCheckout (svn)          Checkout from a svn repository
    setEnv (s)                 Setup development environment (mrbSetEnv)
    build (b)                  Run buildtool
    install (i)                Run buildtool with install
    test (t)                   Run buildtool with tests
    setup_local_products (slp)  Setup local products (mrbslp) [not local sources]
    zapBuild (z)               Delete everything in your build area
    newProduct (p)             Create a new product from scratch
    changelog (c)              Display a changelog for a package
    bumpVersion (bv)           Bump version number of a package
    updateDeps (ud)            Update dependencies in CMakeLists.txt and product_deps
    updateCM (uc)              Update the master CMakeLists.txt file
    makeDeps (md)              Build or update a header level dependency list
    checkDeps (cd)             Check for missing build packages
    pullDeps (pd)              Pull missing build packages into MRB_SOURCE

  Aliases ( we use aliases for these commands because they must be sourced )

   mrbsetenv                  Setup a development enviornment and local products  [use this more often]
                              (source $MRB_DIR/bin/mrbSetEnv)

   mrbslp                     Setup only the products installed in the working localProducts_XXX directory
                              (source $MRB_DIR/bin/setup_local_products)
```

The `mrb` commands are the same if you are on `gm2gpvm` or your Mac.

To initialize your development area, do this in an empty directory.

```
$ mrb newDev
```

```
building development area for gm2 v5_00_00 -q e6:prof

MRB_BUILDDIR is /Users/lyon/Development/g-2/first-try/build_d13.x86_64
MRB_SOURCE is /Users/lyon/Development/g-2/first-try/srcs
INFO: cannot find releaseDB/base_dependency_database
      mrb checkDeps and pullDeps may not have complete information
MRB_PROEJCT IS gm2

IMPORTANT: You must type
    source /Users/lyon/Development/g-2/first-try/localProducts_gm2_v5_00_00_e6_prof/setup
NOW and whenever you log in
```

Read the output carefully. Some things to note:

- A build directory is created and note its name contains the flavor of your machine.[6] You can get to that directory easily with `cd $MRB_BUILDDIR` .

[6] Mac is d13 (for Darwin version 13) and slf5, slf6 are marked as appropriate.

- A source directory is created for your source code. You can get to it easily by doing `cd $MRB_SOURCE` .
- You can ignore the message about the release database. That's a LBNF thing we don't use.
- The important message is indeed important. There is a set up script that you need to run that sets up your environment. Run that script now and whenever you log in to restore your development environment. You don't need to type in the whole path, since you are at the top of your development area.

```
$ source localProducts_gm2_v5_00_00_e6_prof/setup
```

```
MRB_PROJECT=gm2
MRB_PROJECT_VERSION=v5_00_00
MRB_QUALS=e6:prof
MRB_TOP=/Users/lyon/Development/g-2/first-try
MRB_SOURCE=/Users/lyon/Development/g-2/first-try/srcs
MRB_BUILDDIR=/Users/lyon/Development/g-2/first-try/build_d13.x86_64
MRB_INSTALL=/Users/lyon/Development/g-2/first-try/localProducts_gm2_v5_00_00_e6_prof

PRODUCTS=/Users/lyon/Development/g-2/first-try/localProducts_gm2_v5_00_00_e6_prof:/cvmfs/oasis.opensciencegrid.org/gm2/prod/g-2:/cvmfs/oasis.opensciencegrid.org/gm2/prod/external
```

- A local products area is also created. This is your own personal release area that overlays the official one (so stuff you have in your personal release area override products in the official one).

1.3 *Checkout code*

Now you need to checkout some code. For this example, we'll use the `gm2artexamples` product. All of our code lives in `git` repositories on http://redmine.fnal.gov . The `mrb gitcheckout` command is used to clone the git repositories (this is a convenience command so you don't have to remember the git URLs and other set up tasks).[7] Let's check out the `gm2artexamples` product. You must be in the `srcs` directory of your development area. The command is rather chatty.

[7] You can type `mrb g` for short.

```
$ cd srcs
$ mrb g gm2artexamples
```

```
git clone: clone gm2artexamples at /Users/lyon/Development/g-2/first-try/srcs
NOTICE: Running git clone ssh://p-gm2artexamples@cdcvs.fnal.gov/cvs/projects/gm2artexamples
Cloning into 'gm2artexamples'...
X11 forwarding request failed on channel 0
ready to run git flow init for gm2artexamples
Already on 'master'
Your branch is up-to-date with 'origin/master'.
Using default branch names.
Already on 'develop'
Your branch is up-to-date with 'origin/develop'.
Branch develop set up to track remote branch develop from origin.
X11 forwarding request failed on channel 0
Already up-to-date.
NOTICE: Adding gm2artexamples to CMakeLists.txt file
NOTICE: You can now 'cd gm2artexamples'

You are now on the develop branch (check with 'git branch')
To make a new feature, do 'git flow feature start <featureName>'
```

At this moment, you need to switch to a particular feature branch that is compatible with `gm2 v5_00_00`. Do the following,[8]

[8] This step will disappear shortly.

```
$ cd gm2artexamples
$ git flow feature track gm2_5
$ cd ..
```

```
X11 forwarding request failed on channel 0
Switched to a new branch 'feature/gm2_5'
Branch feature/gm2_5 set up to track remote branch feature/gm2_5 from origin.

Summary of actions:
- A new remote tracking branch 'feature/gm2_5' was created
- You are now on branch 'feature/gm2_5'
```

If you have more code to checkout, then run more `mrb g` commands.

1.4 Building code

Now that your code is checked out, you need to build it. The first step you need to do is to "extend" your environment with any products your build depends upon set up. The way to do this is to do `source mrb setEnv`.[9] You need `source` (or `.` for short) because your shell environment needs to be extended with new environment variables. You need to run this command after you log back into and start developing. If you do not make major changes to your code (you don't introduce new dependencies), then you only need to run the command once before you build.

[9] There are two shortcuts for `source mrb setenv`; you can do `. mrb s` or `mrbsetenv` (the latter is a bash function that does the `source` for you).

```
$ . mrb s
```

```
local product directory is /Users/lyon/Development/g-2/first-try/localProducts_gm2_v5_00_00_e6_prof
----------- this block should be empty ------------------
---------------------------------------------------------
ERROR: Cannot do unsetup, SETUP_CETPKGSUPPORT is not defined
The working build directory is /Users/lyon/Development/g-2/first-try/build_d13.x86_64
The source code directory is /Users/lyon/Development/g-2/first-try/srcs
----------- check this block for errors -----------------------
---------------------------------------------------------------
```

For now, ignore the error about `SETUP_CETPKGSUPPORT` (it is benign). You should not see any errors between the dashed lines. If you do, then you have some product dependency mismatch (ask for help).

Now you can build your code. The build command is `mrb build`.[10]

[10] `mrb b` for short

```
$ mrb b
```

The long output is not shown. Hopefully there will be no compilation errors. If you get some, ask for help.

1.5 Testing

`gm2artexamples` is currently the only product that has unit tests. To try them, just do `mrb test`.[11]

[11] `mrb t` for short. A short build check will occur to ensure that everything is built.

```
$ mrb t
```

```
/Users/lyon/Development/g-2/first-try/build_d13.x86_64
calling buildtool -I /Users/lyon/Development/g-2/first-try/localProducts_gm2_v5_00_00_e6_prof -b -t
INFO: Install prefix = /Users/lyon/Development/g-2/first-try/localProducts_gm2_v5_00_00_e6_prof
INFO: CETPKG_TYPE = Prof

------------------------------------
INFO: Stage cmake.
------------------------------------

-- Product is gm2artexamples v2_00_00 e6:prof
-- Module path is /cvmfs/oasis.opensciencegrid.org/gm2/prod/external/art/v1_12_02/Modules;/cvmfs/oasis.opensciencegrid.org/gm2/prod/external/cetbuildtools/v4_03_02/Modules
-- set_install_root: PACKAGE_TOP_DIRECTORY is /Users/lyon/Development/g-2/first-try/srcs/gm2artexamples
-- Building for Darwin d13 x86_64
-- set_install_root: PACKAGE_TOP_DIRECTORY is /Users/lyon/Development/g-2/first-try/srcs/gm2artexamples
-- Selected diagnostics option CAUTIOUS
-- cmake build type set to Prof in directory <top> and below
--    DEFINE (-D): ;NDEBUG
-- compiler flags for directory <top> and below
--    C++     FLAGS: -O3 -g -gdwarf-2 -fno-omit-frame-pointer -Werror -pedantic -std=c++1y -Wall -Werror=return-type
--    C       FLAGS: -O3 -g -gdwarf-2 -fno-omit-frame-pointer -Werror -pedantic   -Wall -Werror=return-type
-- Boost version: 1.56.0
-- Found the following Boost libraries:
--   chrono
--   date_time
--   filesystem
--   graph
--   iostreams
--   locale
--   prg_exec_monitor
--   program_options
--   random
--   regex
--   serialization
--   signals
--   system
--   thread
--   timer
--   unit_test_framework
--   wave
--   wserialization
-- CPACK_PACKAGE_NAME and CPACK_SYSTEM_NAME are gm2artexamples d13-x86_64-e6-prof
-- Configuring done
CMake Warning (dev):
  Policy CMP0042 is not set: MACOSX_RPATH is enabled by default.  Run "cmake
  --help-policy CMP0042" for policy details.  Use the cmake_policy command to
  set the policy and suppress this warning.

  MACOSX_RPATH is not specified for the following targets:

   gm2artexamples_DataObjects_dict
   gm2artexamples_DataObjects_map
   gm2artexamples_HitAndTrackObjects_dict
   gm2artexamples_HitAndTrackObjects_map
   gm2artexamples_Lesson1_HelloWorld1_module
   gm2artexamples_Lesson1_HelloWorld2_module
   gm2artexamples_Lesson1_MyDatumReader_module
   gm2artexamples_Lesson1_ProduceMyLittleDatum_module
   gm2artexamples_Lesson2_makeHits_module
   gm2artexamples_Lesson2_makeRotatedHits_module
   gm2artexamples_Lesson2_makeSimpleTracksFromNewHits_module
   gm2artexamples_Lesson2_makeSimpleTracksFromOldHits_module
   gm2artexamples_Lesson2_readHits_module
   gm2artexamples_Lesson2_readSimpleTracks_module
   test_MyLittleDatumAnalyzer_module
   test_MyLittleDatumProducer_module

This warning is for project developers.  Use -Wno-dev to suppress it.

-- Generating done
-- Build files have been written to: /Users/lyon/Development/g-2/first-try/build_d13.x86_64

------------------------------------
INFO: Stage cmake successful.
------------------------------------


------------------------------------
INFO: gm2artexamples version 2.00.00 configured.
------------------------------------


------------------------------------
INFO: Stage build.
------------------------------------

[  3%] Built target gm2artexamples_DataObjects
[  9%] Built target gm2artexamples_DataObjects_dict
[ 15%] Built target gm2artexamples_DataObjects_map
[ 21%] Built target gm2artexamples_HitAndTrackObjects
[ 28%] Built target gm2artexamples_HitAndTrackObjects_dict
[ 34%] Built target gm2artexamples_HitAndTrackObjects_map
[ 37%] Built target gm2artexamples_Lesson1_HelloWorld1_module
[ 40%] Built target gm2artexamples_Lesson1_HelloWorld2_module
[ 43%] Built target gm2artexamples_Lesson1_MyDatumReader_module
[ 46%] Built target gm2artexamples_Lesson1_ProduceMyLittleDatum_module
[ 50%] Built target gm2artexamples_Lesson2_makeHits_module
[ 53%] Built target gm2artexamples_Lesson2_makeRotatedHits_module
[ 56%] Built target gm2artexamples_Lesson2_makeSimpleTracksFromNewHits_module
[ 59%] Built target gm2artexamples_Lesson2_makeSimpleTracksFromOldHits_module
[ 62%] Built target gm2artexamples_Lesson2_readHits_module
[ 65%] Built target gm2artexamples_Lesson2_readSimpleTracks_module
[ 68%] Built target +Users+lyon+Development+g-2+first-try+build_d13.x86_64+gm2artexamples+bin+myLittleDatum_wr.sh
[ 71%] Built target +Users+lyon+Development+g-2+first-try+build_d13.x86_64+gm2artexamples+bin+very_simple_test.sh
[ 75%] Built target +Users+lyon+Development+g-2+first-try+build_d13.x86_64+gm2artexamples+test+MyLittleDatum_test.d+MyLittleDatum_test.fcl
[ 78%] Built target +Users+lyon+Development+g-2+first-try+build_d13.x86_64+gm2artexamples+test+MyLittleDatum_test.d+messageDefaults.fcl
[ 81%] Built target +Users+lyon+Development+g-2+first-try+build_d13.x86_64+gm2artexamples+test+myLittleDatum_wr.sh.d+MyLittleDatum_r.fcl
[ 84%] Built target +Users+lyon+Development+g-2+first-try+build_d13.x86_64+gm2artexamples+test+myLittleDatum_wr.sh.d+MyLittleDatum_w.fcl
[ 87%] Built target +Users+lyon+Development+g-2+first-try+build_d13.x86_64+gm2artexamples+test+myLittleDatum_wr.sh.d+messageDefaults.fcl
[ 90%] Built target simple_test
[ 93%] Built target test_MyLittleDatumAnalyzer_module
[ 96%] Built target test_MyLittleDatumProducer_module
[100%] Built target test_with_boost

real    0m4.638s
user    0m1.585s
sys 0m1.228s

------------------------------------
INFO: Stage build successful.
------------------------------------


------------------------------------
INFO: Stage test.
------------------------------------

Test project /Users/lyon/Development/g-2/first-try/build_d13.x86_64
    Start 1: very_simple_test.sh
1/5 Test #1: very_simple_test.sh .............   Passed    0.01 sec
    Start 2: simple_test
2/5 Test #2: simple_test .....................   Passed    0.01 sec
    Start 3: test_with_boost
3/5 Test #3: test_with_boost .................   Passed    0.01 sec
    Start 4: MyLittleDatum_test
4/5 Test #4: MyLittleDatum_test ..............   Passed    0.32 sec
    Start 5: myLittleDatum_wr.sh
5/5 Test #5: myLittleDatum_wr.sh .............   Passed    0.76 sec

100% tests passed, 0 tests failed out of 5

Total Test time (real) =   1.12 sec

------------------------------------
INFO: Stage test successful.
------------------------------------
```

1.6 Running

There are several fcl files you can run for `gm2artexamples`.

```
$ ls $MRB_SOURCE/gm2artexamples/fcl
```

```
CMakeLists.txt
hello1.fcl
hello2.fcl
makeAndReadDatum.fcl
makeAndReadTracksFromOldHits.fcl
makeDatum.fcl
makeHits.fcl
makeHitsRotated.fcl
makeTracksFromNewHits.fcl
makeTracksFromOldHits.fcl
messageservice.fcl
minimalMessageService.fcl
readDatum.fcl
readHits.fcl
readSimpleTracks.fcl
```

Our `art` executable is called `gm2`. FCL files are found by the `$FHICL_FILE_PATH` search path.

```
$ gm2 -c hello1.fcl
```

```
%MSG-i MF_INIT_OK:  13-Nov-2014 11:51:41 CST JobSetup
Messagelogger initialization complete.
%MSG
Begin processing the 1st record. run: 1 subRun: 0 event: 1 at 13-Nov-2014 11:51:41 CST
```

```
Hello, world. From analyze. run: 1 subRun: 0 event: 1
Begin processing the 2nd record. run: 1 subRun: 0 event: 2 at 13-Nov-2014 11:51:41 CST
Hello, world. From analyze. run: 1 subRun: 0 event: 2

TrigReport ---------- Event  Summary ------------
TrigReport Events total = 2 passed = 2 failed = 0

TrigReport ------ Modules in End-Path: end_path ------------
TrigReport  Trig Bit#    Visited     Passed     Failed      Error Name
TrigReport     0    0          2          2          0          0 hello

TimeReport ---------- Time  Summary ---[sec]----
TimeReport CPU = 0.000056 Real = 0.000085

Art has completed and will exit with status 0.
```

1.7 Logging in again

At some point, you will want to log out of your machine and log back in later to continue your work. To reconstitute your development environment, you need to,

- Select the release area

  ```
  source /grid/fermiapp/gm2/setup # on gm2gpvm
  source /cvmfs/oasis.opensciencegrid.org/gm2/prod/g-2/setup # On Mac
  ```

- Chnage directory to your development area

  ```
  cd ~/Development/g-2/first-time  # On my Mac
  ```

- Run the setup script in local products (this will re-select the chosen g-2 release)

  ```
  source localProducts_gm2_v5_00_00_e6_prof/setup
  ```

- Extend the environment for the products your build depends upon (don't forget the leading dot)

  ```
  . mrb s
  ```

Now you are set to build (`mrb b`), run (`gm2 -c FCL_FILE`), and develop.

1.8 Summary

Here is a summary of the commands for `gm2 v5_00_00`.

1.8.1 To checkout, build and run `gm2artexmples` to a new development area

```
# Log into machine (e.g. gm2gpvm.fnal.gov)
```

```
# Select release area
source /grid/fermiapp/gm2/setup  # On gm2gpvm
source /cvmfs/oasis.opensciencegrid.org/gm2/prod/g-2/setup # On Mac

# Create development area
mkdir /gm2/app/users/lyon/first-time # For me on gm2gpvm
mkdir ~/Development/g-2/first-time   # For me on my Mac
cd <THAT_DIRECTORY>

# Setup the release
setup gm2 v5_00_00 -q e6:prof

# Initialize Development area
mrb newDev
source localProducts_gm2_v5_00_00_e6_prof/setup

# Checkout code
cd srcs
mrb g gm2artexamples

# Get right branch (for now)
cd gm2artexamples
git flow feature track gm2_5
cd ..

# Extend environment with build dependencies
. mrb s

# Build it
mrb b

# Test it
mrb t

# Run it
gm2 -c hello1.fcl
```

1.8.2 Restoring environment when logging in again later

Here's what you do to restore your environment

```
# Log into machine (e.g. gm2gpvm.fnal.gov)

# Select release area
source /grid/fermiapp/gm2/setup  # On gm2gpvm
source /cvmfs/oasis.opensciencegrid.org/gm2/prod/g-2/setup # On Mac

# cd to development area
cd /gm2/app/users/lyon/first-time # For me on gm2gpvm
cd ~/Development/g-2/first-time   # For me on my Mac

# Restore basic environment
source localProducts_gm2_v5_00_00_e6_prof/setup

# Extend environment with build dependencies
. mrb s

# Now you can work!! For example
mrb b  # Build it if you've made a change since last time
mrb t  # Test it
gm2 -c hello1.fcl # Run it
```

2 Writing Source Code

Your source code lives within a git project checked out to your development area's `srcs` directory. The project has a top level directory[1] that contains the "top level" `CMakeLists.txt` file along with various subdirectories. Code with a common purpose should live in a particular subdirectory.[2] You may mix headers (`.h`, `.hh`), implementation (`.cc`, `.cpp`), and configuration (`.fcl`) files all in the same subdirectory.

[1] For example, the `gm2ringsim` project would get checked out to `srcs/gm2ringsim`, which is the "top level" directory.

[2] Examine `gm2ringsim` for more examples.

2.1 Top level `CMakeLists.txt` file

The top level `CMakeLists.txt` file lives in your top level project directory (e.g. `srcs/gm2ringsim/CMakeLists.txt`). It has the main directives that tells CMake how to build your project.

Below is a representative top level `CMakeLists.txt` file.[3] The `mrb newProduct` command will create a skeleton file for you.

[3] There are five main parts of the file (roughly in order in the file)...

- Defining the project
- Loading CMake macros and setting the CMake environment
- Setting compiler options
- Specifying external packages that will be used
- Specifying subdirectories that contain a `CMakeLists.txt` file and, perhaps, code to build

```
# Ensure we are using a moden version of CMake
CMAKE_MINIMUM_REQUIRED (VERSION 2.8)

# Project name - use all lowercase
PROJECT (gm2analyses)

# Define Module search path
set( CETBUILDTOOLS_VERSION $ENV{CETBUILDTOOLS_VERSION} )
if( NOT CETBUILDTOOLS_VERSION )
  message( FATAL_ERROR
         "ERROR:␣setup␣cetbuildtools␣to␣get␣the␣cmake␣modules" )
endif()
set( CMAKE_MODULE_PATH $ENV{CETBUILDTOOLS_DIR}/Modules
                                         ${CMAKE_MODULE_PATH} )

# art contains cmake modules that we use
set( ART_VERSION $ENV{ART_VERSION} )
if( NOT ART_VERSION )
  message( FATAL_ERROR
         "ERROR:␣setup␣art␣to␣get␣the␣cmake␣modules" )
```

```
endif()
set( CMAKE_MODULE_PATH $ENV{ART_DIR}/Modules
                                          ${CMAKE_MODULE_PATH} )

# Import the necessary macros
include(CetCMakeEnv)
include(BuildPlugins)
include(ArtMake)
include(FindUpsGeant4)

# Configure the cmake environment
cet_cmake_env()

# Set compiler flags
cet_set_compiler_flags( DIAGS VIGILANT WERROR
   EXTRA_FLAGS -pedantic
   EXTRA_CXX_FLAGS -std=c++11
)

cet_report_compiler_flags()

# Set include and library search paths (the version numbers
# are minimum -  if actual version of product is below specified,
# will get error)

# Everyone should include these
find_ups_product(cetbuildtools v3_07_08)
find_ups_product(art v1_08_10 )
find_ups_product(fhiclcpp v2_17_12)
find_ups_product(messagefacility v1_10_26)

# This project uses code from gm2ringsim,
# gm2dataproducts, and gm2geom
find_ups_product(gm2ringsim v1_00_00)
find_ups_product(gm2dataproducts v1_00_00)
find_ups_product(gm2geom v1_00_00)

# This project uses code from Root
find_ups_root(v5_34_12)

# Make sure we have gcc
cet_check_gcc()

# Macros for art_make and simple plugins (must go after
# find_ups lines)
include(ArtDictionary)

# Specify subdirectories to build
add_subdirectory( ups )  # Every project needs a ups subdirectory
add_subdirectory( DisplayDataProducts )
add_subdirectory( calo )
add_subdirectory( fcl )
add_subdirectory( test )
add_subdirectory( util )

# Packaging facility - required for deployment
include(UseCPack)
```

2.1.1 When you need to add/change a line in top level `CMakeLists.txt`

There are two situations for which you will have to alter the top level `CMakeLists.txt` file:

If you add, delete, or rename a subdirectory If you add a subdirectory, you must write a corresponding `add_subdirectory(dirName)` directive.[4] If you delete a directory, you must remove its corresponding `add_subdirectory` line. If you rename a directory, you must edit its corresponding `add_subdirectory` line to reflect the change. If you do not follow these steps, then some code may not build (without an error, so this mistake will be hard to find) or you may get an error when CMake tries to build a directory that no longer exists.

[4] The `add_subdirectory` directory tells CMake to go into that subdirectory and build code there. If you don't have the `add_subdirectory` then CMake won't look in the subdirectory at all.

You use code from an external project If you use code from an external project, you may need to add a corresponding `find_ups_product` or similar line.[5]

[5] See section 2.8 for instructions.

2.2 Organizing Source Code

The build system we use is quite flexible and you can organize your code in many ways. You may be used to having all of your header files in an `include` directory with the `.cc` files in other directories. This artificial separation is unnecessary. You may group files together any way you like and may have header files and implementation files in the same directory. Typically, it is best to group files by topic or functionality.

2.3 Writing Modules

Modules are plugins to art that perform certain functions (analyzers, producers, filters, and output modules). See section 10 of the Art Work Book[6] for more information. Only reminders will be given here.

[6]

You should use `artmod` to write the skeleton of the module. Do `artmod --help-types` to see the list of module types it will make. Then just run it, giving the name of the class you want including any namespace specification. For example,

```
artmod producer tracking:TrackFinder
artmod analyzer gm2analysis::CalorimeterDiags
```

Remember that you specify the class name, not the file name (so do not give `_module` in the name).

2.4 *Writing Services*

TODO

2.5 *Writing Input Source Modules*

TODO

2.6 *Directory level `CMakeLists.txt` file*

If your subdirectory (e.g. `srcs/gm2analyses/strawTracker`) has anything to build, has header files, or has further subdirectories, then it must have a `CMakeLists.txt` file (and a corresponding `add_subdirectory` line in the `CMakeLists.txt` from the directory above - see Sec. 2.1.1).[7] If your subdirectory has code to build, then the directory `CMakeLists.txt` file needs to have

[7] The directory level `CMakeLists.txt` file is different from the top level `CMakeLists.txt` file. The latter is in your project top level directory, like `srcs/gm2analyses`. The former is in a subdirectory of that top level and is described in this section.

```
art_make(  )
```

A directory with no `.cc` or `.cpp` files has no code to build and so does not get an `art_make` line in the directory `CMakeLists.txt` file.

See the next section (Sec. 2.6.1) for arguments to the `art_make`. You should call `art_make` only once per `CMakeLists.txt` file.

If your subdirectory has header files, then those have to be copied to the release area when one runs `mrb install`. To do that, you need a line the directory `CMakeLists.txt` file with

```
install_headers( )  # No arguments
```

If your subdirectory has fcl files, then those need to be copied to the build area as well as the release area. There is some scripting involved to do that (put the following in the directory `CMakeLists.txt` file),

```
# install all *.fcl files in this directory to the release area
file(GLOB fcl_files *.fcl)
install( FILES ${fcl_files}
         DESTINATION ${product}/${version}/fcl )

# Also install to the build area
foreach(aFile ${fcl_files})
  get_filename_component( basename ${aFile} NAME )
  configure_file(
         ${aFile} ${CMAKE_BINARY_DIR}/${product}/fcl/${basename}
         COPYONLY )
endforeach(aFile)
```

If your subdirectory has futher subdirectories with code to build, then you need an `add_subdirectory( dirName )` line for each subdirectory.

2.6.1 Arguments to `art_make`

You can find documentation for `art_make` in its source code at `$ART_DIR/Modules/ArtMake.cmake`. Basically, you need to specify what libraries to link against when you use external code.[8] If you don't use any external code, then you will have no arguments to `art_make`. It will tell CMake to build all regular source, modules, services, and input sources in the directory. If you do use external code, then you have four choices,

[8] See Sec. 2.8 for how to tell if you are using external code.

- If the source file using external code is a regular source (not a module, not a service, not an import source), then you need

```
    art_make(
          LIB_LIBRARIES
          library1
          library2    # if needed
    )
```

- If the source file using the external code is a module source (e.g. `analyze_my_hits_module.cpp`) then you need

```
    art_make(
          MODULE_LIBRARIES
          library1
          library2    # if needed
      )
```

- If the source file using the external code is a service source (e.g. `analyze_my_hits_service.cpp`) then you need

```
    art_make(
          SERVICE_LIBRARIES
          library1
          library2    # if needed
      )
```

- If the source file using the external code is source code for an input source
(e.g. `midas_source.cpp`) then you need

```
    art_make(
            SOURCE_LIBRARIES
            library1
            library2   # if needed
       )
```

If you have a mixture of sources in your directory, you can string the calls together. For example,[9]

```
    art_make (
        LIB_LIBRARIES
          ${ROOT_GPAD}
        MODULE_LIBRARIES
          gm2analyses_util
          gm2analyses_strawtracker_util
         )
```

[9] In the example to the left, regular sources get linked against Root's `libGpad.so` (see Sec. 2.8.2) and modules get linked against code built in the `srcs/gm2analyses/util` and `srcs/gm2analyses/strawtracker/util` directories (see Secs. 2.8.4 and 2.8.5).

Note that it does not hurt for code to build against a library that it doesn't need. So if you have five modules and only one needs to link against a library, put that library in the `MODULE_LIBRARIES` section. The one that needs it will link against it and the four that don't won't care.

2.7 *Libraries produced from building*

Every directory in your project that has code to build generates at least one library.[10] Say, for example, you have a directory called `gm2analyses/calo`. Regular sources (not modules, services, nor input sources) get compiled and the objects go into a library called `libgm2analyses_calo.so` (the name is the directory path with slashes replaced by underscores). Each module in the directory gets its own library. For example, if there is a module in that directory called `Analyze_Calo_module.cc` then that code will go into a library called `libgm2analyses_calo_Analyze_Calo_module.so`. A similar thing happens for services and input sources. Therefore, one directory of code may produce several libraries. The `art_make` directive in the directory `CMakeLists.txt` file tells the build system to build code and make the corresponding libraries.

[10] An important note, if your directory **only** has header files in it (should be a rare situation for code written by users), then no library will be produced (because there is no code to build - the header files are all included by other source code). You still need the directory level `CMakeLists.txt` file for the `install_headers()` directive, but do not do `art_make`. See Sec. 2.6.

2.8 *Using External Code (Linking)*

Your code is almost never self-contained, especially when writing within the Art framework. You may use functions and classes from external libraries, like Root and Geant4. You may use algorithms, data products, and other functionalities from other projects, like gm2ringsim. You may use objects defined in other directories in your project. If you are writing an art module or service, you may use objects defined in the same directory, but in a different file from the module or service. All of these examples are "external code".

Art uses *dynamic linking*, which means that the art executable (ours is called gm2) has very little code in it. Instead, it loads all of the libraries it needs at runtime. The other style is *static linking* where the executable has embedded in it all of the libraries it needs. Dynamic linking, as the name suggests, allows for flexibility with one executable able to load a variety of different libraries decided upon at runtime with the configuration file. There is, however, overhead in dynamic loading typically experienced as slow start-up time of the program. Static linking produces an executable with all of the libraries built in - so there is little flexibility in terms of functionality. But the start up time is much faster. Static linking typically leads to many copies of executables for the different functionalities, resulting in duplication of libraries that are in common. For maximum flexibility and non-duplication of libraries, art loads everything dynamically.

HOW DO YOU KNOW WHEN YOU ARE USING EXTERNAL CODE? An easy indicator is when you have a #include for a header file. For each #include, you need to think and perhaps add a corresponding link directive in a CMakeLists.txt file.[11] If you forget to link to a library that you need, you will get a missing symbol error when you try to run. This section will explain how to figure out these situations and actions you need to take.

[11] Remember the two types of CMakeLists.txt files: "top level" and "directory level". The former (see Sec. 2.1) is the potentially big file at the top level of your project. The latter (see Sec. 2.6) is the smaller file in the directory with your actual source code files.

2.8.1 Includes for system headers and base art headers

System headers, like #include <string> do not require any special directives for linking. You get them for free.

Headers in art, fhiclcpp, and messagefacility do not require anything in your directory level CMakeLists.txt file. The corresponding libraries are automatically loaded by the art executable. Your top level CMakeLists.txt file must contain the following lines,[12]

```
...
cet_report_compiler_flags()
...
find_ups_product(art v1_08_10 )
find_ups_product(fhiclcpp v2_17_12)
find_ups_product(messagefacility v1_10_26)
...
```

[12] These lines add header file directories to the compiler include search path (e.g. without them, you will get a compilation error that header files cannot be found).

2.8.2 Includes for Root headers

Including a header from Root is a little unusual because you do not have to give a path in the include, e.g. `#include "TCanvas.h"` (not `#include "root/TCanvas.h"`). If you include a header from Root, you will also need to link to the corresponding Root library. First, in the top level `CMakeLists.txt` file, you must have,[13]

```
...
cet_report_compiler_flags()
...
find_ups_root(v5_34_12)
...
```

[13] That `find_ups_root` line adds the Root headers to the compiler include search path and creates CMake variables corresponding to each Root library.

If you look at the code for the `find_ups_root` CMake macro at `$CETBUILDTOOLS/Modules/FindUpsRoot.cmake` you will see lines like,[14]

```
find_library(ROOT_GLEW NAMES GLEW PATHS ${ROOTSYS}/lib
                    NO_DEFAULT_PATH)
find_library(ROOT_GPAD NAMES Gpad PATHS ${ROOTSYS}/lib
                    NO_DEFAULT_PATH)
find_library(ROOT_GRAF NAMES Graf PATHS ${ROOTSYS}/lib
                    NO_DEFAULT_PATH)
find_library(ROOT_GRAF3D NAMES Graf3d PATHS ${ROOTSYS}/lib
                    NO_DEFAULT_PATH)
```

[14] These lines define the CMake variables that correspond to Root libraries. You use them in the directory level `CMakeLists.txt` file to tell CMake to link against that library.

To determine the Root library you need, look up the Root object in the Root documentation at http://root.cern.ch/drupal/content/reference-guide (select the appropriate version of Root - usually the PRO version). Find the class name from the list and click on it. On the new page, on the very right hand side in a little greyed out box it will say the library that corresponds to that Root object. For example, if you `#include "TCanvas.h"` you need to link against the `libGpad` library. The CMake variable name will in general be the name of the library, all upper case, with the `lib` replaced by `ROOT_`. So `libGpad` → `${ROOT_GPAD}`.

In your directory level `CMakeLists.txt` file, you will have the `art_make` directive. Add the appropriate CMake variable corresponding to the Root library you need. See Sec. 2.6.1 for where to put such items in the arguments. For example,[15]

```
      art_make (
         LIB_LIBRARIES
            ${ROOT_GPAD}
         MODULE_LIBRARIES
            ${ROOT_TREE}
            ${ROOT_TVMA}
```

[15] In the example left, regular sources are linked against `libGpad.so` while modules are linked against `libTree.so` and `libTVMA.so`.

```
        )
```

2.8.3 Includes for GEANT headers

To include a header file from Geant4, requires you to have `Geant4/` in the header path, for example `#include "Geant4/G4Track.hh"`. If you include such headers in your code, then you will also need to link against the Geant4 libraries. First, in your top level `CMakeLists.txt` file, you must have,

```
...
cet_report_compiler_flags()
...
find_ups_geant4(v4_9_6_p02)
...
```

That line adds the Geant4 headers to the compiler include search path and creates the CMake variables `${G4_LIB_LIST}` and `${XERCESLIB}`. For any Geant4 header, just add those CMake variables to the `art_make` directive in your directory `CMakeLists.txt` file. See Sec. 2.6.1 for where to put such items in the arguments. For example,
`srcs/gm2ringsim/calo/CMakeLists.txt` has, in part,[16]

```
art_make(
    LIB_LIBRARIES
        gm2geom_calo
        gm2geom_station
        artg4_material
        artg4_util
        ${XERCESCLIB}
        ${G4_LIB_LIST}
    SERVICE_LIBRARIES
        gm2ringsim_calo
      )
```

[16] If you are curious, you can see where `G4_LIB_LIST` is defined in `$CETBUILDTOOLS_DIR/Modules/FindUpsGeant4.cmake`. `XERCESLIB` goes with Geant.

2.8.4 Includes for headers in the project

The `#include` directive should include the path to the header file, including the name of the project even if the header is in the same directory as the source, though you could just give the header file name. For example, if `CaloHitSD.hh` is in the `gm2ringsim/calo` directory, then `CaloHitSD.cc`, when it includes `CaloHitSD.hh`, can do either

```
#include "CaloHitSD.hh"
```

or

```
#include "gm2ringsim/calo/CaloHitSD.hh"
```

The latter is preferred as it is clearer, but if you change the name of the directory, you must change the include as well.

If you have a regular source file and it includes a header that is present in the same directory, then you do not need to do anything to the `CMakeLists.txt` files. If you have a module, service, or input source file and it includes a header that is present in the same directory, then you need to link against the library for that directory. You do not need to add anything to the top level `CMakeLists.txt` file. To the directory `CMakeLists.txt` file, you must add the library. See Sec. 2.6.1 for where to put such items in the arguments. For example, `srcs/gm2ringsim/calo/CMakeLists.txt` has, in part,[17]

```
 art_make(
     LIB_LIBRARIES
         gm2geom_calo
         gm2geom_station
         gm2ringsim_station
         artg4_material
         artg4_util
         ${XERCESCLIB}
         ${G4_LIB_LIST}
     SERVICE_LIBRARIES
         gm2ringsim_calo
       )
```

[17] In the left example, services in that directory are linked against the library that gets created from the regular sources, namely `libgm2ringsim_calo.so`. You can predict the name of the library by taking the source directory (e.g. `gm2ringsim/calo`) and replacing the slashes by underscores.

If any source file uses a header that present in a different directory in your project, then you must link against that library. In the example above, code in the `gm2ringsim/calo` directory includes code from `gm2ringsim/station`, and hence `gm2ringsim_station` is present in the arguments of `art_make`.

An important exception to these instructions is if the directory with the header file contains **only** header files. In that case, that directory produces no libraries and you do not have to change the directory `CMakeLists.txt` file.

2.8.5 Includes for headers in other projects

If you have a source file (regular, module, service, or input source) that uses code from another project, then you need to do some work. An example here is code in `gm2ringsim` uses code from the `gm2geom` and `artg4` projects. The `#include` needs the path to the header file including project name, directory name and header name. For example, `#include "artg4/util/util.hh"`.

In your top level `CMakeLists.txt` file, you need a `find_ups_product` line for the project specifying the project name and a minimum version number. See Sec.2.1 for an example.

In your directory `CMakeLists.txt` file, you need to list the library corresponding to the code you are using. See Sec. 2.6.1 for where to put such items in the `art_make` arguments. For example, `srcs/gm2ringsim/calo/CMakeLists.txt` has, in part,

```
art_make(
    LIB_LIBRARIES
        gm2geom_calo
        gm2geom_station
        artg4_material
        artg4_util
        ${XERCESCLIB}
        ${G4_LIB_LIST}
    SERVICE_LIBRARIES
        gm2ringsim_calo
      )
```

When the regular sources are built, they will be linked against code in `gm2geom/calo`, `gm2geom/station`, `artg4/material`, and `artg4/util`.

An important exception to these instructions is if the directory with the header file contains **only** header files. In that case, that directory produces no libraries and you do not have to change the directory `CMakeLists.txt` file. You still need to have the top level `CMakeLists.txt` file correct as described above.

3

Common Things You Do...

This chapter contains some reminders of common things you do in Muon $g-2$ code.

3.1 Dealing with parameters

The constructor for your module or service has the parameter set as an argument. You can retrieve information from the parameter set and supply defaults if the parameter does not exist as in the example below.

```
gm2ex::CalorimeterDigitizer::CalorimeterDigitizer(
               fhicl::ParameterSet const & p) :
  category_    (p.get<std::string>("category","digi")),
  TAURAMP_     (p.get<float>("TAURAMP", 1.4 /* ns */)),
  TAUDECAY_    (p.get<float>("TAUDECAY", 36.4 /* ns */)),
  PULSELENGTH_ (p.get<int>("PULSELENGTH", 30 /* samples */)),
// ...
```

3.2 Readling enviornment variables

```
 #include <cstdlib>
//...
std::string value = std::getenv("PATH'');
```

The argument to `std::getenv` is a constant character array, not a `std::string`.

3.3 Throwing an exception

See http://mu2e.fnal.gov/public/hep/computing/exceptions.shtml.

```
#include "cetlib/exception.h"
// ...
if ( something ) {
  throw cet::exception(CATEGORY) << "Message\n"
}
```

3.4 Finding a file

`cetlib` has a nice facility for searching for files in a path specification. See `$CETLIB_INC/cetlib/search_path.h`.

It may be convenient to specify the search path in a FHICL parameter with the possibility of providing an environment variable. Here is some code that takes a search path through the parameter, but if the first character is a $, it then gets the path through the specified environment variable.

```
gm2util::MetadataFromFile::MetadataFromFile(
          fhicl::ParameterSet const & p) :
    searchPath_  (p.get<std::string>("searchPath", ".")),
    fileName_    (p.get<std::string>("fileName")),
    keyName_     (p.get<std::string>("keyName"))
{
  // Let's parse the search path
  // If the first character is a dollar sign, then the
  // remaining is an environment variable
  if ( searchPath_.at(0) == "$" ) {
    std::string envVar = searchPath_.substr(1);
    char* envValue = std::getenv(envVar.c_str());
    if ( ! envValue ) {
      searchPath_ = ".";
      throw cet::exception("META_DATA_FROM_FILE") <<
       "Environment variable " << envVar << " is not set";
    }

    searchPath_ = std::string(envValue);
  }
}
```

Index